Development of a GUI RDF tool for generating XML/RDF graph model

Aditya Deepak

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Development of a GUI RDF tool for generating XML/RDF graph model

Thesis submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Aditya Deepak

(Roll: 111CS0066)

under the guidance of

Prof. Dr. S.K.Rath

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

March' 2015

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India.

May 9, 2015

Certificate

This is to certify that the work in the thesis entitled "***Development of a RDF tool for generating XML/RDF graph model***" by ***Aditya Deepak*** is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Dr. Santanu Kumar Rath
Professor

Acknowledgment

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis.

Foremost, I would also like to express my gratitude towards my project advisor, Prof. Santanu Kumar Rath, whose mentor-ship has been paramount, not only in carrying out the research for this thesis, but also in developing long-term goals for my career. His guidance has been unique and delightful. He provided his able guidance whenever I needed it. I would also like to thank my Ph.D. mentor Mr. Ashish Kumar Dwivedi who helped me greatly by being a source of knowledge to me.

I would also like to extend special thanks to my project review panel for their time and attention to detail. The constructive feedback received has been keenly instrumental in improvising my work further.

My parents receive my deepest love for being the strength in me.

Aditya Deepak

Roll No. 111CS0066

Abstract

With the rapid increase in the usage of Representational State Transfer Architecture (REST) for the web-services, it becomes mandatory for the proper representation of data and their dependence on each other for the fruitful use of the dataset for various processes operating at disjoint independent locations on the web. The work presented here mainly focuses on building a tool for visual RDF and OWL editor for representing information of various data nodes in a RDF graph model and the inter-relationships with respect to each other and deploying the knowledge bases on it for the auto-generation of the XML files for transfer of data across the Semantic web.

Keywords:[Representational State Transfer(REST), Web Ontology Language(OWL), Resource Description Framework(RDF), Extended Markup Language(XML), Command Line Interface(CLI), Strongly Connected Component(SCC), Friend of a Friend Ontology(FOAF), Application Programming Interface(API)].

Contents

Certificate ii

Acknowledgement iii

Abstract iv

List of Figures vii

1 Introduction 1

1.1 The Problem Statement . . . 1

1.2 Entity-Relationship Model . . . 2

1.3 The Semantic Web . . . 3

2 Literature Review 5

2.1 RDF Work . . . 5

3 Proposed Work 6

3.1 How to Partition Data . . . 6

3.2 RDF Tool Modeling . . . 6

3.3 Merging Ontologies . . . 7

3.4 Used and Proposed Algorithms . . . 8

4 Simulation and Results 11

4.1 Experimental Setup . . . 11

4.2 Dataset Collection . . . 11

4.3 Testing . . . 12

4.4 Comparison . . . 12

4.5 XML file generation from the RDF Tool . . . 14

4.5.1 Input from GUI . . . 14

4.5.2 Output XML file from the RDF model 15
4.5.3 Deletion of the nodes in the GUI tool 16
4.6 Merging of Ontologies 17
4.6.1 Input RDF model to JENA API 17
4.6.2 Query Result returned by Jena API 18
5 Conclusion **22**
Bibliography **23**

List of Figures

1.1 Entity Relationship Model 2
1.2 Stack of the Semantic Web 4

4.1 Existing Algorithm Result with intermediate steps 13
4.2 Single DFS Proposed Algorithm Result 14
4.3 The GUI RDF tool from where we can design RDF files 15
4.4 The XML/RDF file generated from the graphical tool 16
4.5 Deletion of resources 17
4.6 After deletion of nodes new dumped XML file 18
4.7 Intersection region of two ontologies 19
4.8 Input to the Jena API 20
4.9 Output from the Jena API 21

Chapter 1

Introduction

Presently the amount of data in the World Wide Web (WWW) is extremely enormous and is growing day by day. Research is going on to come up with techniques, standards, languages etc., for the proper representation, manipulation and organization of those data. The Resource Description Framework (RDF) is a framework for representing enormous information on the web and the relationships between them. The information can be maintained as either document-store, key-value or graph database. The RDF itself is a graph database but with an abstraction, contrary to it in the inner working, it is a collection of forests which itself are trees: the information on a particular website is stored in a hierarchical form so that the retrieval of the required information is with ease and little overhead.

1.1 The Problem Statement

Since the amount of data is increasing enormously; there is a need for proper manipulation and representation of the data so that the required information can be processed with little effort. For this a relational entity model was proposed by Peter P.Chen[1]. The E-R model represents the data in an abstract way, where the resources are inter-connected to each other with predicate connections and this model can be represented in RDF model for storing the information in the Semantic Web, which helps the user to analyze the schema and the relationships in a RDF model. In the last decade a new way of representing data came into existence, i.e. the Semantic Web. This provides opportunities to the cluster of machines to process the data stored

in World Wide Web. In addition to it representing the information in RDF model is a herculian task, for this a GUI RDF tool is required to solve the problem with little effort.

1.2 Entity-Relationship Model

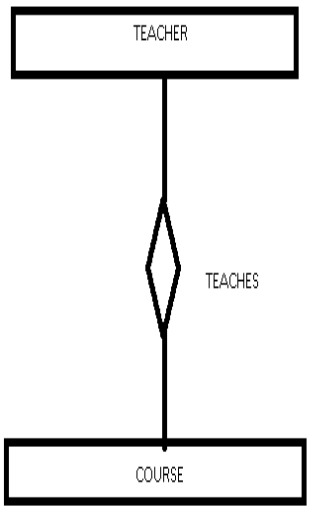

Figure 1.1: Entity Relationship Model

The Entity-Relationship model can be divide into three parts: the relationship set, the entity set, and the value set. An entity is something that has its separate existence, but in E-R model it may happen that some entities are sub-classes of another entity. Relationships are the connection between the two entities. Relationships can be of the three types itself: one-one (1:1), one-many (1:m), and many-many (n:m). Figure 1.1 shows a many to many relationship between the entities Teacher and Courses.

1.3 The Semantic Web

The Semantic Web was presented by the founder of World Wide Web. The term Semantic Web is presented in [2]. Semantic Web work is to make the contents of WWW accessible and readable for a machine. This means that the structure of the web page should be dynamic and also a lot of meta-data should be present in the web page. The two technologies that help to achieve the same are: XML and RDF. XML allows the users to make their own tags provided what they are doing to exploit a particular feature of the XML format. Users can add arbitrary structures to their documents. RDF is more structured oriented, which itself is expressed through XML format. But it has its own standards.

The Semantic Web helps the machines to understand and respond to the request made by humans based on their meaning (ontology). For such an understanding by the machines requires the relevant information to be stored as RDF triples. An example of tag in nonsemantic webpage is <item> Blog </item>, where as in a semantic webpage it looks like as <Subject> <Predicate> <Object> , comprising the triple format of the RDF.

In figure 1.2 the lowest level comprises of the URI which act as the source of resources for all the data. Above it works the XML layer, where the data are encoded in the XML format and meta data are embedded inside it to make the XML/RDF graph model. Over the XML layer connections between resources are built by giving them the proper relationships, and then over the built ontology, queries are ran with the help of SPARQL language and appropriate results are obtained.

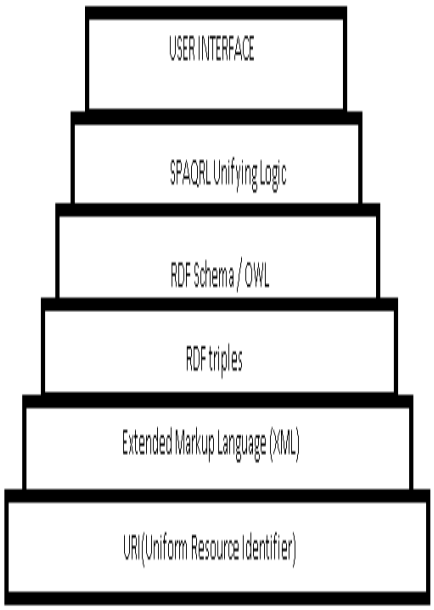

Figure 1.2: Stack of the Semantic Web

Chapter 2

Literature Review

2.1 RDF Work

From the last decade Resource Description Framework (RDF) has been a popular area of research work. Several researchers have worked on accurate and efficient retrieval of information from the RDF and encoding of the knowledge, derived from the inferences drawn by the Web Ontology Language (OWL) and also to encode the knowledge into the existing Ontology. Some notable contributions in this area are: T.Berners Lee [2], Jiawen Huang [3]. There is a huge increase in the data over the past decade which demands for the partition of the data across various cluster to achieve the required performance while querying the data. In clustered RDF database systems the number of researches made are less. The currently available ones , such as SHARD[4], YARS2[5], generally hash partition algorithm which distributes the RDF triples across multiple machines, and work parallely to access these machines as much as possible during query execution.

There has been a significant progress in the research effort of building high performance data management system and for the ontology. This started with early work on Jena[4], and RDF Suite[5]. But unfortunately as the amount of RDF data is increasing day by day, it is not possible to store the whole data on a single machine and still able to access the required data with high performance. Therefore clustered RDF databases are coming into the picture.

Chapter 3

Proposed Work

3.1 How to Partition Data

Whenever the amount of data is huge partitioning of the data across various machines and also looking for efficient retrieval algorithm during the query operation performed on the RDF dataset is of utmost importance. Partitioning the data using the hashing algorithm is best to perform task in parallel. So partitioning the data with respect to RDF triple's subject then there is a high probability that all triples of RDF are stored on the same machine. Hence,every machine in the cluster can process the queries parallely and the results can be aggregated together across various machines at the end of the query firing.

If the complete triple is not required to be queried in the query fired by the end user then shuffling of data will be mandatory at query time, and if there is no need for a particular pattern of the query, the intermediate results puts burden on the Jena reasoner. Hence instead of chosing hash partitioning concept, we use Graph Partition RDF algorithm based on the SCC concept. So the vertices which are close to RDF graph machine need to be stored at the same machine and graph patterns can be queried parallely thus reducing the time of the query.

3.2 RDF Tool Modeling

To design a GUI tool for XML file checking and generation of GUI based XML tree/RDF model. It takes a XML file input and checks for well-formed structured of the XML file. If there is any error in the file, it gives the line number and

column number of the error. Implemented BFS and DFS for offline query in the tree (Document Object Model) DOM. It generates the GUI based XML tree of the input XML file in java. The tree has features:

1. Getting the content of a node by clicking on it.
2. Adding a node at the leaf node of the tree.
3. Adding a node in between two nodes.
4. Updating the content of the node.
5. Deleting a node from the tree.
6. Deleting the subtree of the tree.
7. Swapping two subtrees.
8. After each change, the new xml file is dumped in the xml file and the new XML tree is displayed. This plays an importat role for the RDF generation rather than doing it from CLI its all done with Graphical User Interface.

3.3 Merging Ontologies

After the RDF data set is made from the RDF tool, integration of the meta-data into the RDF model for querying and drawing inferences from the ontology in necessary. Merging of the ontologies is done by loading the named RDF graph models into the APACHE Jena API and with the help of SPARQL query, conclusion can be drawn by executing the queries of the end users and caching the appropriate results obtained from the RDF data model for further processing. Merging of different named RDF graphs by loading it into the Jena API creates a graph model in which the unique resources are connected to each other by a strong relation so the required results can be extracted with less effort. While merging, the Jena model internally draws a graph model and establishes the links between the various resources in the form of predicate linking two resources. Here instead of an undirected graph, the model is represented in the form of a directed graph.

3.4 Used and Proposed Algorithms

In the existing algorithm to partition the data across various clusters the entry time into a particular resource and also the exit time is required, for which $Node.discovery[node]$ and $Node.lowest[node]$ are kept. Here $Node.lowest[node]$ represents whether there is a back edge in the modeled graph. After obtaining the exit time of each and every resource sort the exit time in non-increasing order and push all the required resource which are visited in the path, into the stack $STK[]$. Then do the transpose of the input graph model and then run the same $dfsTranspose(Node)$ function recursively in the transpose graph obtained by popping the resources out of the stack $STK[]$ and caching the results in the main memory and later using it.

In the Single DFS proposed algorithm, transpose graph model is not required, here simply ran the $dfs(node)$ search function recursively and stored the visited resources in the stack $STK[]$. Then performed a topological sorting on the given directed graph model and obtained the dependencies between the resources which are linked. While exploring the dependencies, parallely ran the reasoner of the Jena API and obtained the required results. As the transpose graph is not required the space complexity gets reduced by half of the existing algorithm, and a second $dfs(node)$ is not necessary to run on the modeled graph so reducing the complexity of the proposed algorithm.

Algorithm 1: Existing Algorithm

Require: $Node.discovery$[node],$Node.lowest$[node]
node: key of the input i.e. one url/uri
$neighbour - list[]$: All the urls that can be visited from the current key url
$STK[]$: Stack Data-structure
$visited[]$: to check if a node if visited or not
$start - time[]$: start-time of the node
$end - time[]$: end-time of the node
time : global counter to track the time
count : to count number of groups formed
$Transpose - list[]$: Transpose graph of the input RDF graph model

```
1: for node in neighbour-list and neighbour-list[!visited[node]] do
2:    goto 11
3:    count <− count+1
4: end for
5: sort(finish.begin(),finish.end())
6: reverse(finish)
7: for finish[node] do
8:    goto 19
9:    count <− count+1
10: end for
11: dfsNew(node)
12: start[node.u] <− time
13: time <− time+1
14: visited[node.u] <− true
15: for !visited[neighbour-list[node.u]] do
16:    goto 11
17: end for
18: finish[node.u] <- time
19: time <- time+1
20: dfsTranspose(node)
21: visited[node.u] <- true
22: STK.push(node.u)
23: for !visited[Transpose-list[node.u]] do
24:    if !visited[Transpose-list[node.u]] then
25:       goto 19
26:    else
27:       STK.pop() until STK.top != node.u
28:    end if
29: end for
```

Algorithm 2: Single DFS Proposed Algorithm

Require: $Node.discovery$[node],$Node.lowest$[node]
node: key of the input i.e. one url/uri
$neighbour - list$[]: All the urls that can be visited from the current key url
STK[] : Stack Data-structure
visited[] : to check if a node if visited or not
time : global counter to track the time
count : to count number of groups formed

1: **for** node in neighbour-list **do**
2: goto 5
3: count < − count+1
4: **end for**
5: dfs(node)
6: Node.discovery[node] < − time
7: Node.lowest[node] < − time
8: time < − time+1
9: push node into STK[]
10: When no neighbour of node is there :
do:
Pop all nodes from STK
Until node!=STK.top()
goto 11
goto 3
11: **if** !visited[neighbour-list[node]] **then**
12: Node.lowest[node] < − min(Node.lowest[node.u],Node.lowest[node.v])
13: Node.lowest[node] < − min(Node.lowest[node.u],Node.lowest[node.v])
14: **end if**

Chapter 4

Simulation and Results

4.1 Experimental Setup

1. Java
2. JDK 7
3. Ubuntu and Windows-8
4. Virtual machine

4.2 Dataset Collection

A spider was designed with scrapy in python to scrap data from the amazon website for the books and the HC-BIOGRID for the medical dataset.

Scraped Information

Amazon Book Crossing	43854
HC-Biogrid	43854
...	

4.3 Testing

Sample Data

input

page	nodes	edges
0	4089	43854
1	4089	43854

output

page	group-count
0	2023
1	895

In the input table *nodes* is the number of unique resources in the RDF graph model and *edges* is the total number of unique edges (predicate) connecting the resources. In the output table $group - count$ is the number of Strongly Connected Components which are obtained by running both the algorithms.

4.4 Comparison

Space Comparison

1. On closely observing the graph model, the RDF model can be represented in the form of adjacency-list/adjacency-matrix. If represention is done in adjacency-list format less space is required and computation can be done in linear memory O(n),but if represented in the adjacency-matrix format it requires more space O(n*n).

Time Comparison

Existing Algorithm :
V: number of nodes(resources), E: number of edges(relationships)

1. Time Complexity : O(n) = K*O(V+E+VlogV), K = constant assumed 2 and is loop invariant

2. Runtime : 0.64 second

Single DFS Proposed Algorithm :

V: number of nodes(resources), E: number of edges(relationships)

1. Time Complexity : O(n) = K*O(V+E), K = constant factor which is loop invariant k = 1

2. Runtime : 0.34 seconds

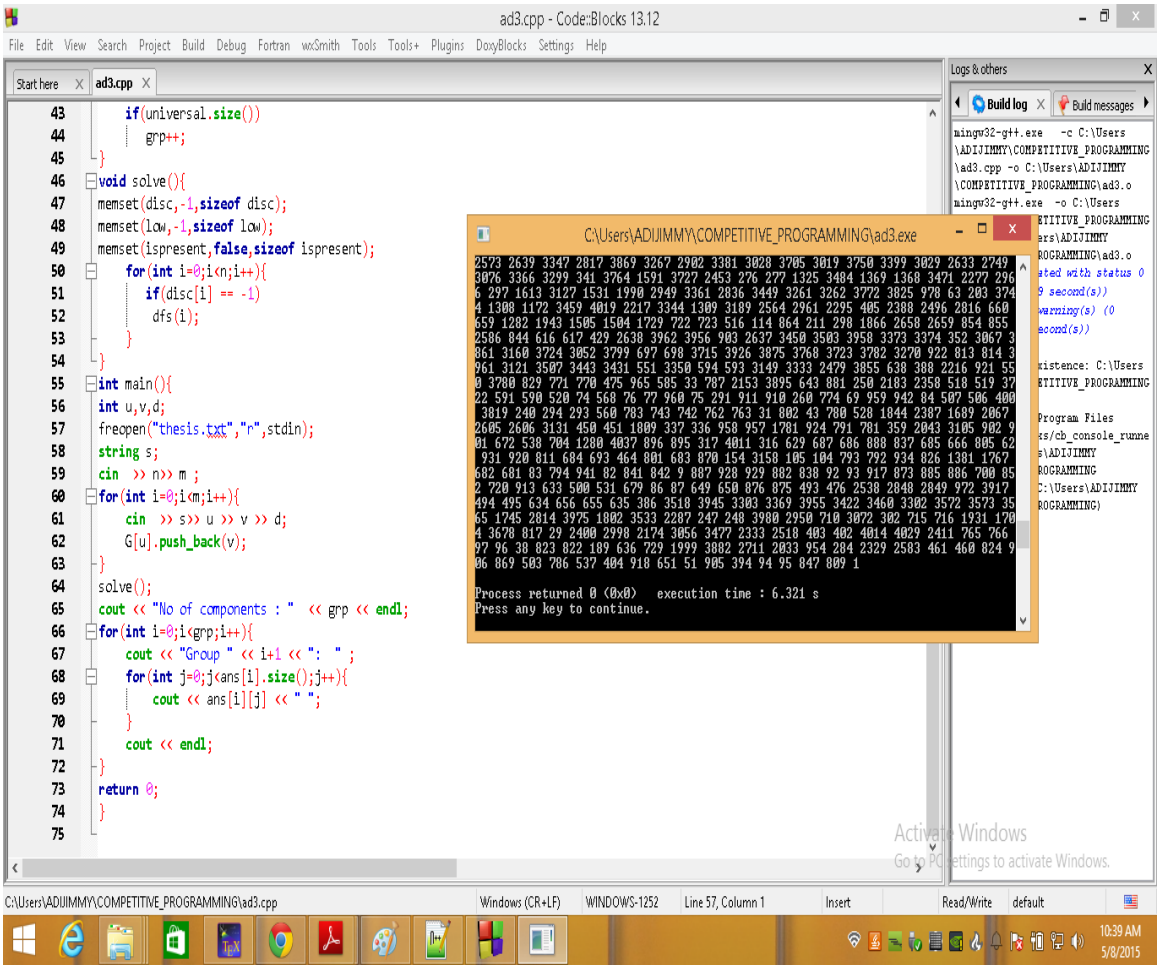

Figure 4.1: Existing Algorithm Result with intermediate steps

The Single DFS Proposed algorithm performs better in terms of time complexity and the space complexity solely depends on the representation of the data, than existing algorithm because in the existing algorithm sorting of resources with respect to the finish time is required, which introduces extra time complexity of O(V logV), but this complexity is not required in case of the Single DFS Proposed algorithm.

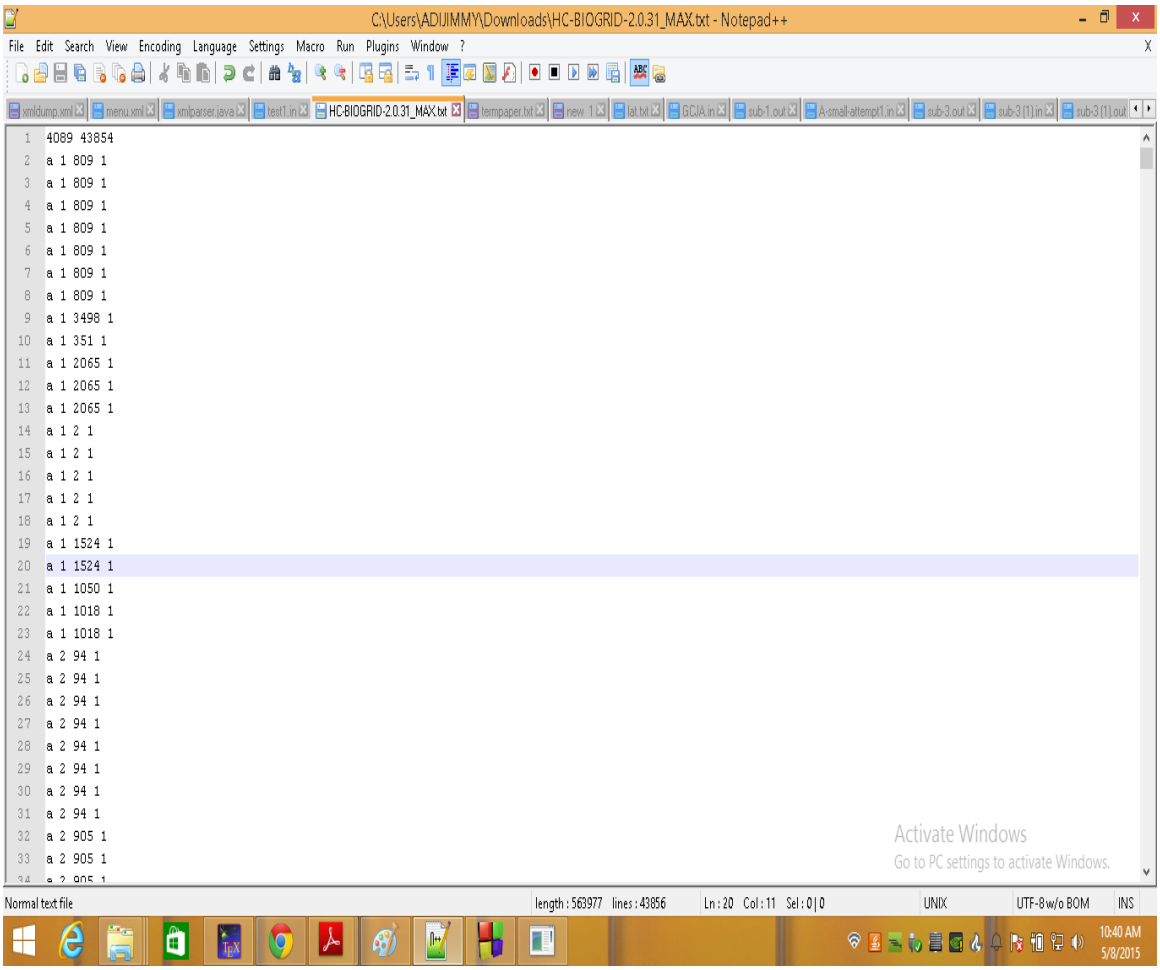

Figure 4.2: Single DFS Proposed Algorithm Result

4.5 XML file generation from the RDF Tool

4.5.1 Input from GUI

Figure 4.3 is the snapshot of the RDF tool generated with the JAVA language where at level 0, there is the root node and all other information are encoded into the RDF graph model in a hierarchical manner. On clicking a particluar class details of that particular class/subclass are displayed. After the design of the RDF data model is over the generated class logic model is dumped into an XML file, which can be later used by the Jena API to query for the results using SPARQL query language.

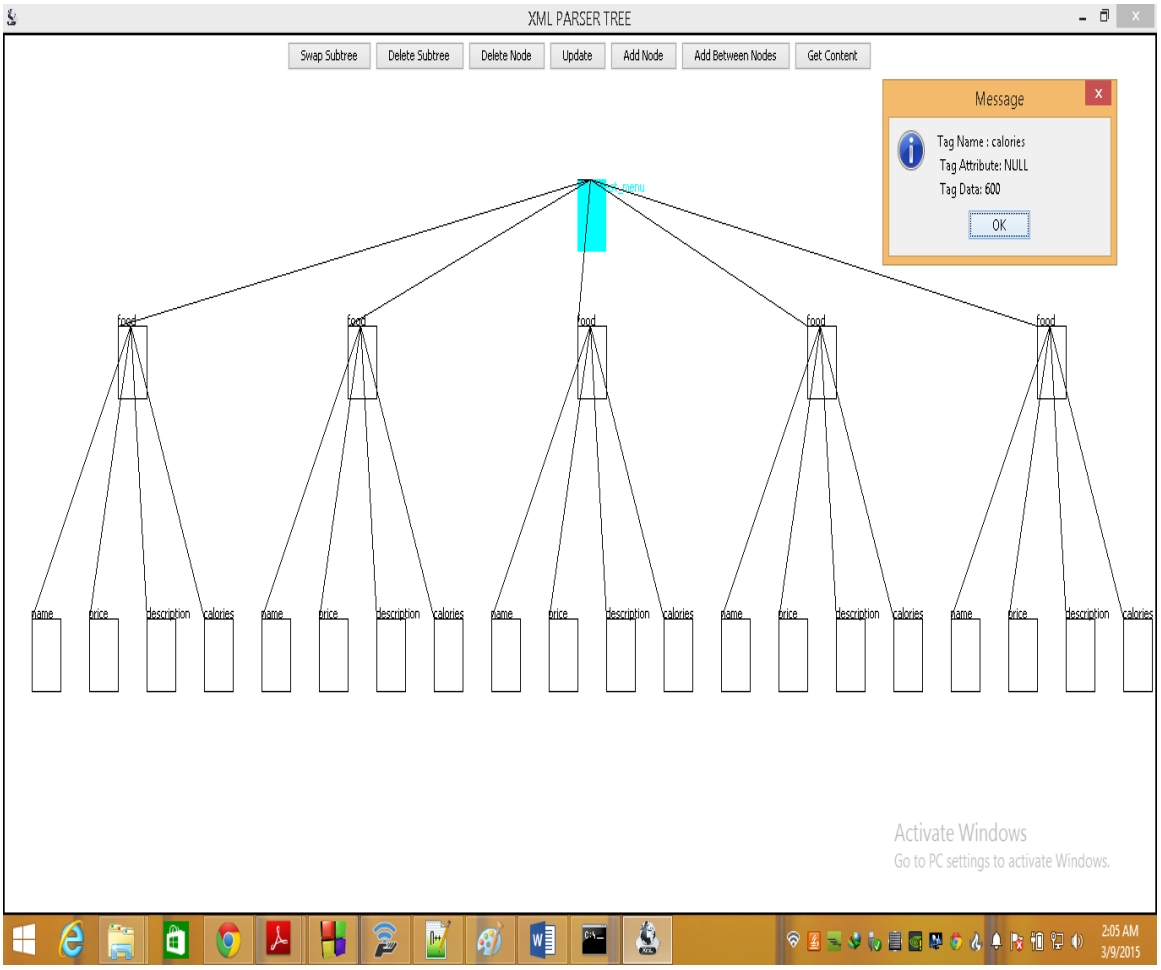

Figure 4.3: The GUI RDF tool from where we can design RDF files

4.5.2 Output XML file from the RDF model

Figure 4.4 is the snapshot of the XML/RDF file obtained from the RDF generation GUI tool, after classes, subclasses and triple relations are given to the tool it is automatically dumped into the output XML file, and the reverse can also be accomplished. The tool is capable of even reading a XML file and show the required relation in a graphical form in the tool interface. If the input format is not correct the tool is also capable of raising exception and let the developer know where there is a problem in the XML file.

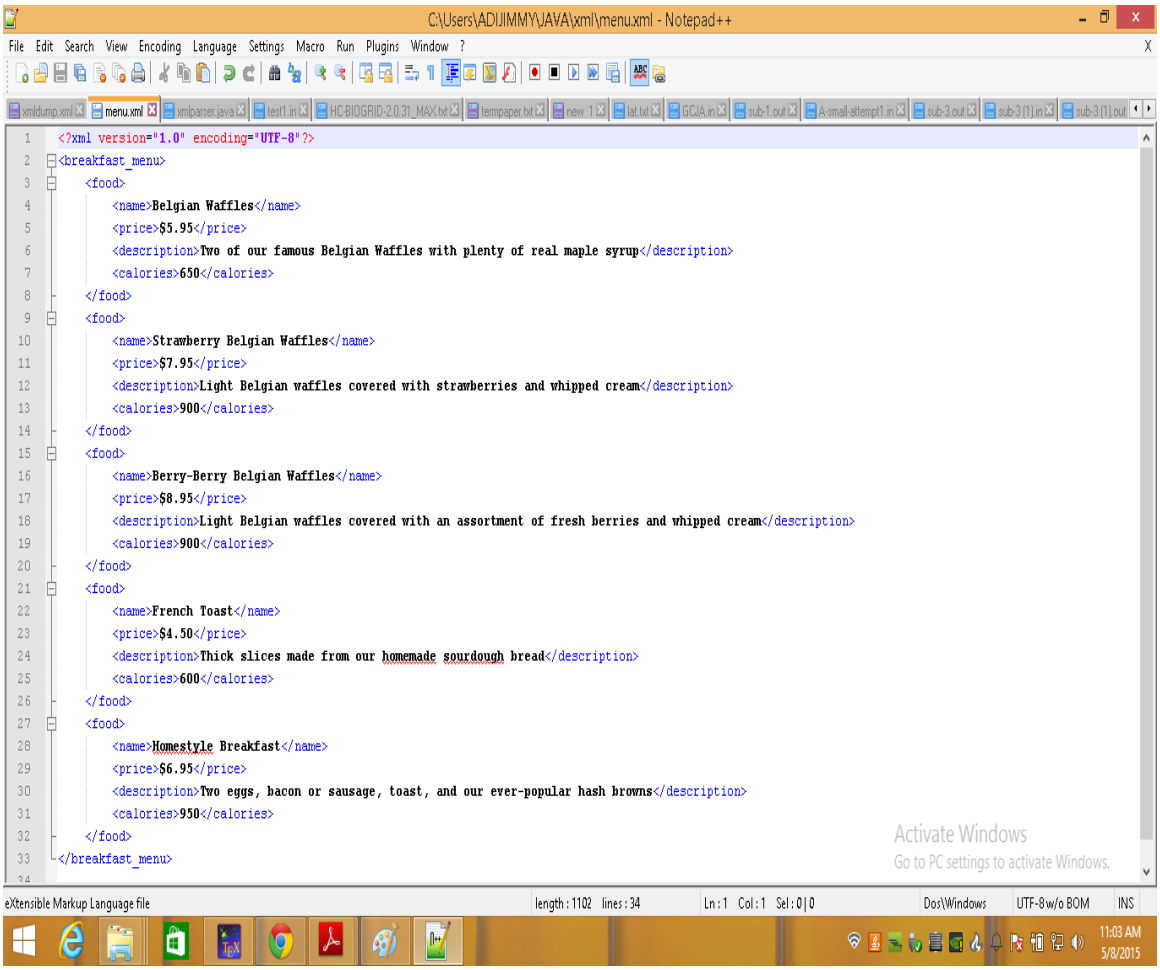

Figure 4.4: The XML/RDF file generated from the graphical tool

4.5.3 Deletion of the nodes in the GUI tool

Figure 4.5 is the snapshot of GUI RDF tool where the nodes from the third child of the root node is deleted and all of its sub-children so the resultant RDF model looks like the figure 4.5.

New dumped look of the XML file

Whenever there is a change in the GUI for re-designing the RDF model it is simultaneously communicated to the XML file which takes the output of the RDF tool at every iteration of the change made to the model.

Hence after the deletion of some nodes as demonstration the dumped XML looks like figure 4.6.

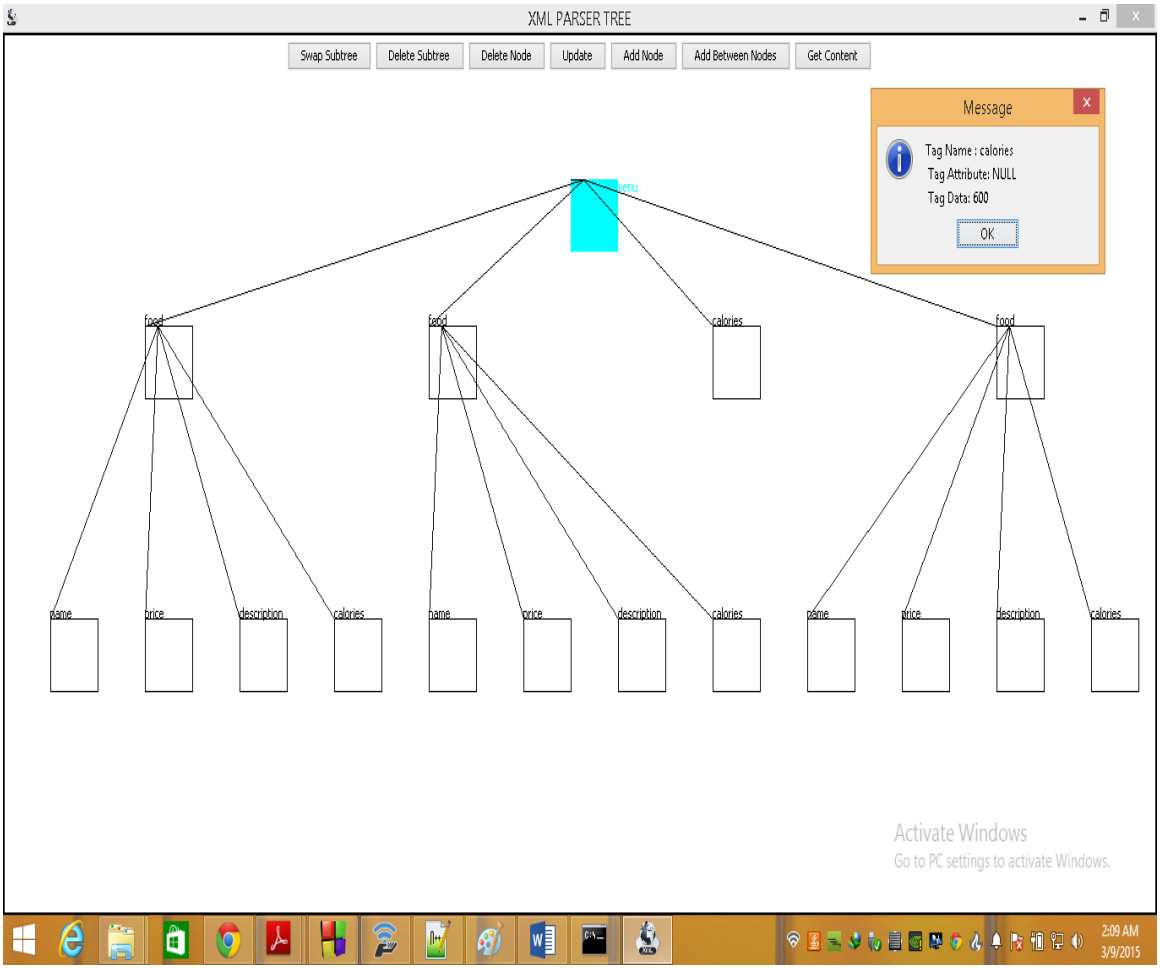

Figure 4.5: Deletion of resources

4.6 Merging of Ontologies

The concept of merging the ontologies, after naming the graph models which are required to be loaded into the Jena API for reasoning between the intersection region of the ontologies, and drawing conclusions by the Jena API reasoner is shown in Figure 4.7

4.6.1 Input RDF model to JENA API

The input model to the Jena API is written using the JAVA programming language specifying the required models, which needs to be loaded into the Java Virtual Machine for the query reasoner to work on it. Figure 4.8 shows how the model is loaded into the memory.

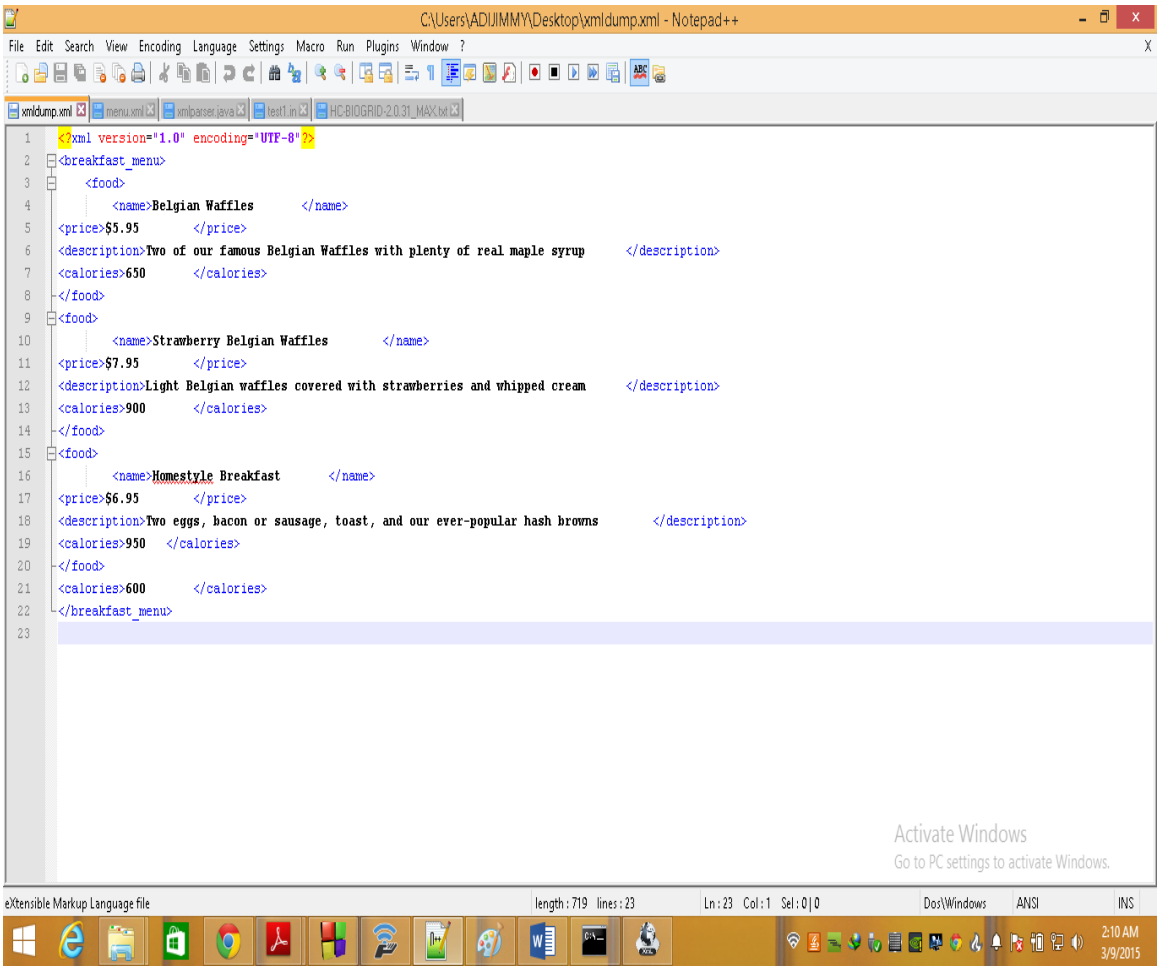

Figure 4.6: After deletion of nodes new dumped XML file

4.6.2 Query Result returned by Jena API

The query result is returned by the reasoner of the Jena API as required by the user, even it is possible to obtain the result in many formats available, which it can be sent across the Semantic Web for further processing. The results can be generated in XML/RDF, TURTLE, JSON formats.

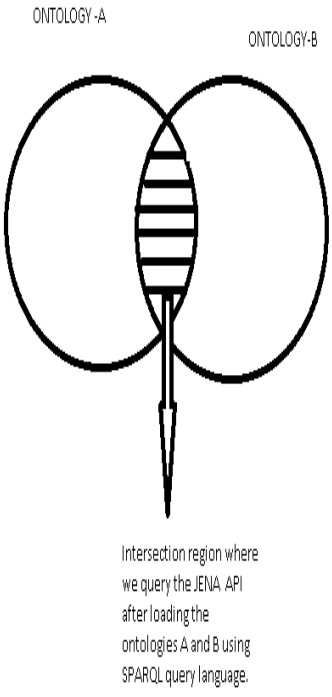

Figure 4.7: Intersection region of two ontologies

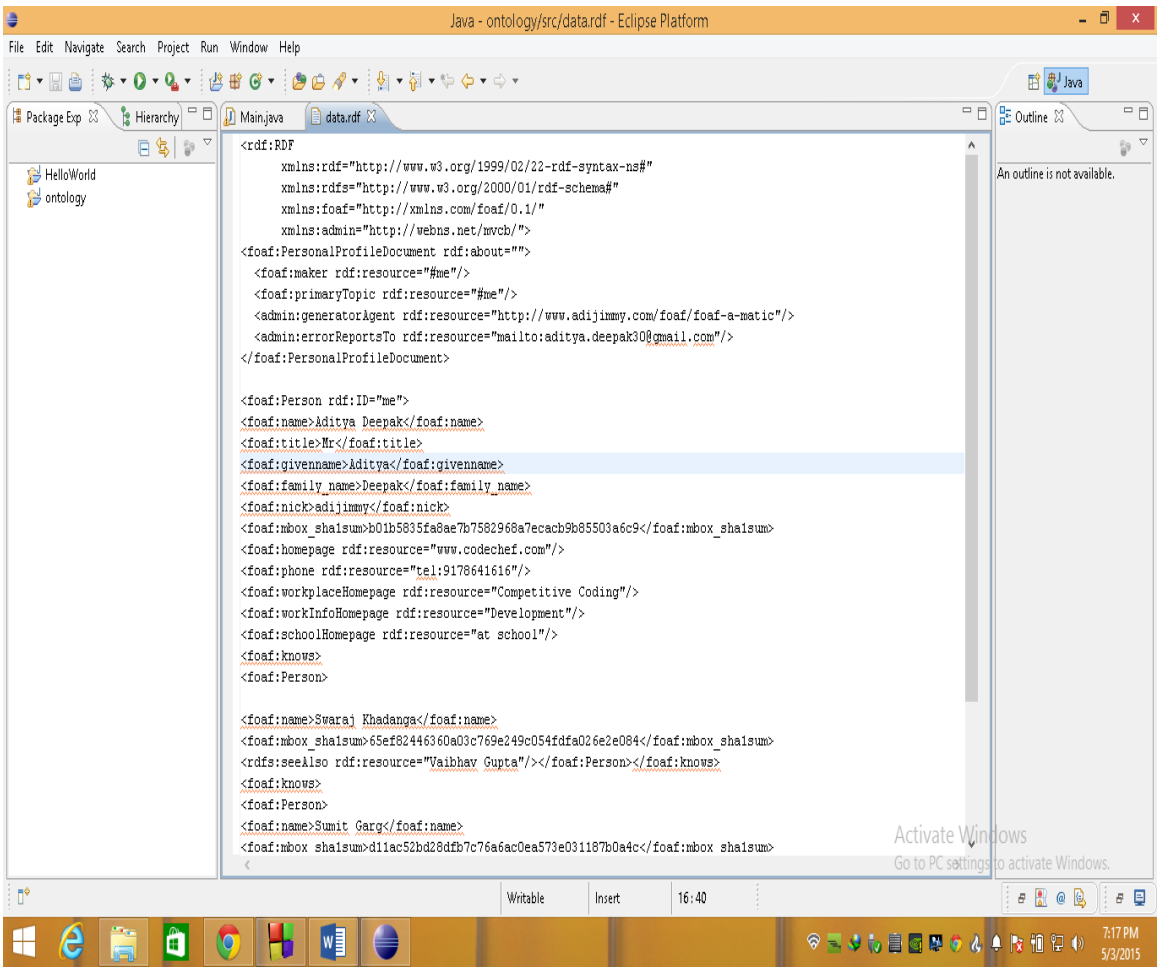

Figure 4.8: Input to the Jena API

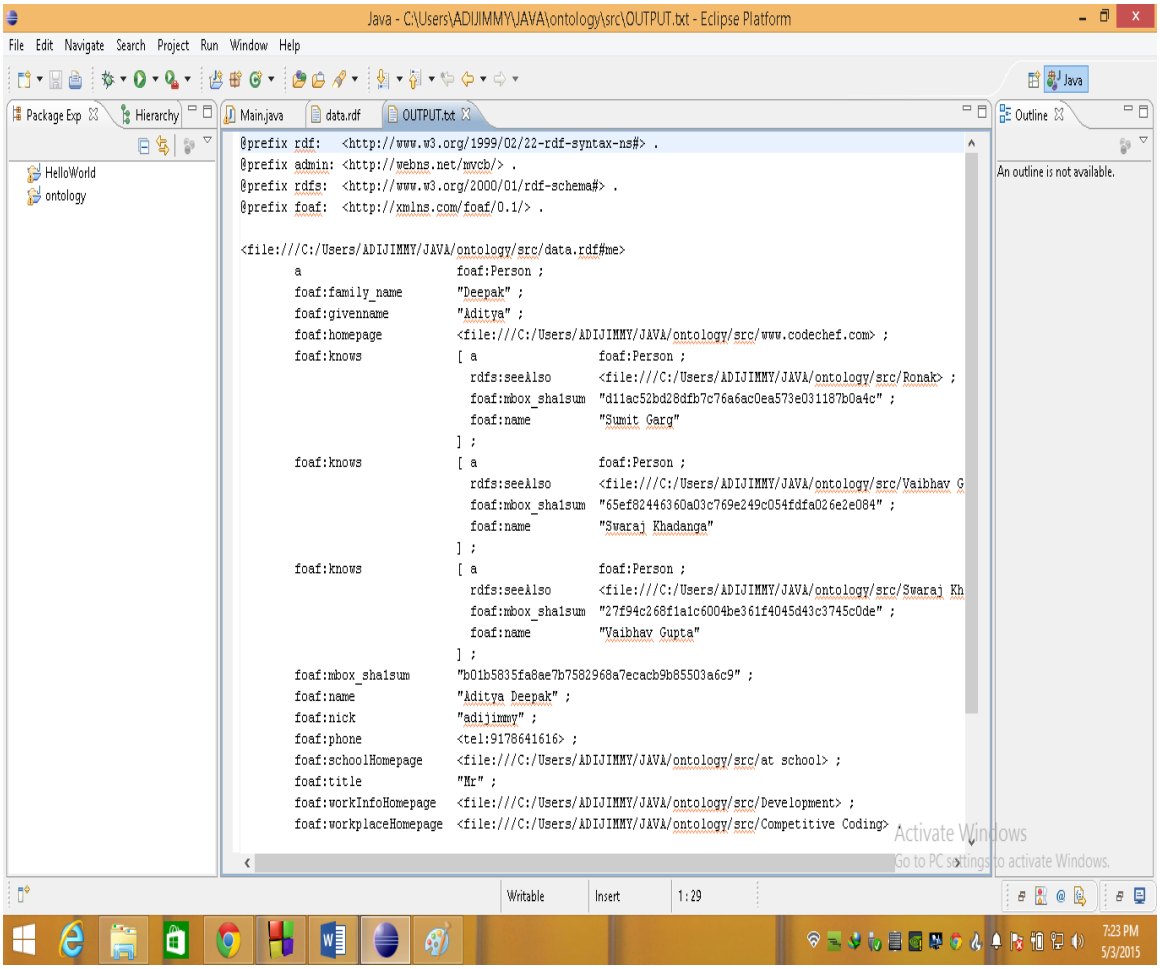

Figure 4.9: Output from the Jena API

Chapter 5

Conclusion

Proper partitioning of data across machines helps to reduce the amount of time required to query the RDF graph model. For the partitioning of data, Single DFS proposed algorithm was used and tested on the large data-set of Amazon BookStore and HC-BIOGRID data.

Ontology plays an important role in the Semantic Web world by embedding the required knowledge into the RDF graph model. A graphical tool was designed to represent the RDF graph model and for testing, queries were made on FOAF ontology, and the results obtained were validated using the graph model made by the GUI RDF tool.

Bibliography

[1] Peter Pin-Shan Chen. The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[2] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[3] Jiewen Huang, Daniel J Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[4] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ICS-FORTH RDFSuite: Managing voluminous RDF description bases. In *SemWeb*, 2001.

[5] Kevin Wilkinson, Craig Sayers, Harumi A Kuno, Dave Reynolds, et al. Efficient RDF storage and retrieval in Jena2. In *SWDB*, volume 3, pages 131–150. Citeseer, 2003.